CHALMERS

Automatic generation of OpenDrive roads from road measurements

Master of Science Thesis in the Programme Computer Science-Algorithms, Languages, Logic

HAN SHI

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, November 2011

Automatic generation of OpenDrive roads from road measurements

Han Shi

shhan@student.chalmers.se

Examiner: Graham Kemp

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

The picture shows the inside of the simulator at VTI, There is a big screen in front of the car. One can drive on the road that displayed on the screen. The road is generated by OpenDRIVE file. It is described in detail at page 4, section 2.2.

Department of Computer Science and Engineering
Göteborg, Sweden November 2011

Abstract

VTI, Swedish National Road and Transport Research Institute is an independent, research institute in the transport sector. Human Behaviour Analysis in the transport system is VTI's most important responsibility. VTI is developing auto driving system, which needs road information for the simulator. The only data format that the simulator supported is OpenDRIVE data format, unfortunately we do not have an existing data source which is OpenDRIVE data format. Therefore one needs to convert the other kind of GPS data source to OpenDRIVE data format. Then GPS data from OpenStreetMap has been chosen as the source of this project. What the program needs to do is tracing road that described in OpenStreetMap data and converting map information to OpenDRIVE data format. The result is reasonable and good enough so that VTI could use this way and program to generate OpenDRIVE file that they want.

Contents

Figures

Tables

1.Introduction

1.1 Background

National Road and Transport Research Institute (VTI) is an independent research institute in the transport sector. Human Behaviour Analysis in the transport system is VTI's most important responsibility. Driving simulators offer a unique opportunity to study the interaction between man and machine. VTI have used simulators since the 1970s.

VTI uses a standard format which is called OpenDRIVE to describe and generate paths including road, highway, motorway, river etc. in their driving simulators. This style uses a path following coordinate system, i.e. Non-linear.

OpenDRIVE is an open format specification that is used to describe traffic information including paths tracing and road network's logic in certain area. The main objective of OpenDRIVE is to standardize the traffic description in order to make conversion between different driving simulators convenient (1). It is a XML format specification which was developed and maintained by a team of simulation professionals with large support from the simulation industry.

Today there are a lot of map data available in electronic form, which should be used, to generate both roads and surrounding landscape. However, since only OpenDRIVE data format file could be used in the simulator at VTI, so other kinds of map data format should be converted to OpenDRIVE before it is used. Unfortunately it is not an easy job to do the conversion between two different kinds of data formats, basically it requires good algorithms for converting from linear coordinates to another coordinate system.

1.2 Objectives

The main purpose of this thesis is generating OpenDRIVE roads automatically based on the data sources, so I need to choose the most suitable data source and convert it to OpenDRIVE data format.

The simulator needs to generate roads in the real world, so we need to have some GPS data source which could give us information of those roads. Firstly we need to decide which kind of data source shall be used in the project, here we use data source from OpenStreetMap which provides free geographic data of xml or png version to anyone who wants them. OpenStreetMap is a free editable map of the whole world. One can export .xml file which contains information of all the roads within choosing area. The road in OpenStreetMap.xml file is described with a series of consecutive nodes, and each node has its unique location in the map.

However, the road in OpenDRIVE.xml file is described with a series of geometries, it separates a long road into several segments, and each segment can be described with a geometry. There are four kinds of geometries that OpenDRIVE could support: Line, Spiral, Arc, Cubic polynomials, we decide to describe all the segments with cubic polynomials, i.e. $y = a + b*x + c*x^2 + d*x^3$ (the reason will be mentioned in 1.4). But the road in OpenStreetMap is described with nodes, so we need to fit those nodes into geometries.

Based on what I said before, the main purposes of this thesis are:

- Choose suitable GPS data source for the project (OpenStreetMap data source is selected here).

- Separate the road into several segments by a reasonable way.

- Construct curves that has the best fit to a series of nodes in OpenStreetMap for each segment

- Generate OpenDRIVE format file with chosen data source.

1.3 Limitations

Some road features that defined in OpenDRIVE format file like “elevationProfile” (elevation of road), “crossfall”(cross fall of road) need information of altitude, i.e. three dimensional coordinates system. But until now, all the GPS data sources that I could get are two dimensional, including the data sources that is using in the project, so the project cannot implement elevation of the road with the current data sources. We cannot generate landscape also for the same reason.

2.Background

This chapter goes through some of the concepts and techniques that readers need to know to read the rest of the report.

- Least squares: Least squares method is used to do curve fitting, i.e. plot geometry with nodes described in OpenStreetMap.
- OpenDRIVE: the only data format that could be used in the simulator, the target file that the project wants to generate.
- OpenStreetMap: GPS data source that provides road information.
- Bézier Curves: another possible solution that could be used to do curve fitting.
- Coordinates systems: these three coordinates systems would be used during the conversion process.

2.1 Least squares

"The method of least squares is a standard approach to the approximate solution of over determined systems" (2). Least squares method can solve the problem when there are some unknowns that you need to calculate based on some given equations(number of equation must be no less than unknown). The main goal of least squares method is adjusting the parameters of a model function to fit a given data set and minimizing the sum of squared error simultaneously. Error here means the difference between actual value and the fitted value that calculated by the model. Besides constructing model, it is also used to estimate the error with which parameters were calculated.

Here is an simple problem statement of Least squares problem: A given n nodes data set: $\{(x_i, y_i), i = 1,2,...n\}$, x_i is independent variable and y_i is dependent variable. An unknown model function f(x, v), v is a vector which consists of all the parameters that should be calculated. The objective is to set all the parameters to fit the nodes in the give data set. The least squares method can find the optimum solution by minimizing squared errors:

$$E = \sum_{i=1}^{n} e_i^2$$

$$e_i = y_i - f(x_i, v)$$

Least squares problems can be separated into two categories: linear least square and non-linear least squares. What the project mentioned is the linear least-squares problem. A problem is a linear least squares problem when the parameters of model function are linear, i.e.

$$f(x_i, v) = \sum_{j=1}^{m} v_i \varphi_j(x_i)$$

φ_j are the coefficients of functions of x_i, it is the partial derivative of model function:

$$\varphi_j(x_i) = \frac{\partial f(x_i, v)}{\partial v_i}$$

Let all the partial derivatives be zero, we can get m linear equations, substitute them into model function f(x, v), we will get all the parameters that we want.

Least squares could also be separated into partial least squares and total least squares. “Partial least squares regression (PLS regression) is a statistical method that bears some relation to principal components regression; instead of finding hyperplanes of maximum variance between the response and independent variables, it finds a linear regression model by projecting the predicted variables and the observable variables to a new space”(3). “Total least squares regression is a least squares data modeling technique in which observational errors on both dependent and independent variables are taken into account” (4). Figure 2.1 shows the difference between total least squares and partial least squares

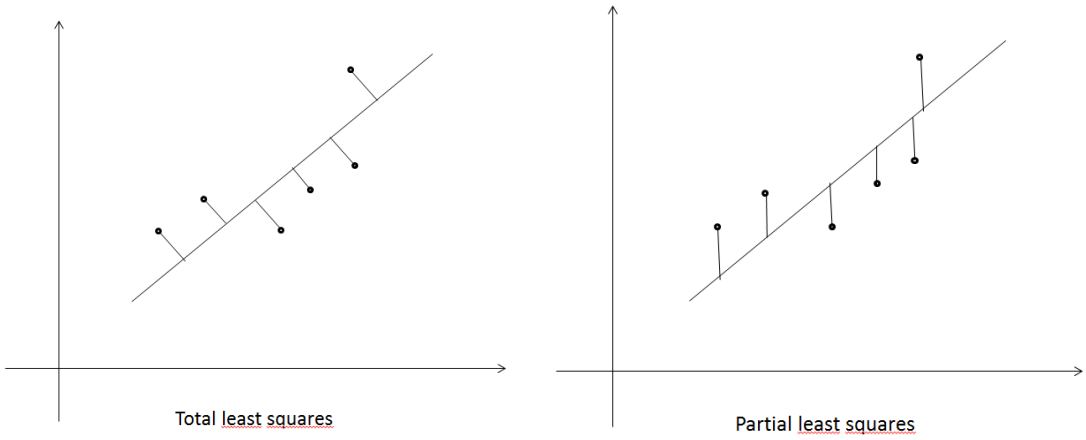

Figure 2.1 difference between total least squares and partial least squares

As shown in the figure, the main difference between total least squares and partial least squares is the mode of error evaluation, error in total least squares is defined as the absolute distance between node and curve, yet error in partial least squares is

defined as the distance between node and curve in y-axes(5). Obviously the error in total least squares is more precise, so what the project uses is total least squares, the error mentioned in the report is always based on total least squares.

2.2 OpenDRIVE

OpenDRIVE was developed as an interface of the databases with auto driving traffic, since there are many simulators which use different kinds of data format, so VIRES(an acronym for Virtual Reality and Simulation) developed a way of standardizing the road description to facilitate the data exchange between different simulators(6).

The OpenDRIVE file format provides the following features:

"XML format

hierarchical structure

analytical definition of road geometry(plane elements, elevation, crossfall, lane width etc.)

various types of lanes

junctions incl. priorities

logical inter-connection of lanes

signs and signals incl. dependencies

signal controllers (e.g. for junctions)

road surface properties

road and road-side objects

user-definable data beads" (7)

Each OpenDRIVE file describes the transportation services in a certain area. Here is a simple example of OpenDRIVE data format and comments explain attributes that the project mentioned is shown in figure 2.2:

```
<OpenDRIVE>
<header
revMajor="1"
revMinor="1"
name="Testfile"
version="1"
date="Thu Feb 8 14:24:06 2007"
north="2.0000000000000000e+003"
south="-2.0000000000000000e+003"
east="2.0000000000000000e+003"
west="-2.0000000000000000e+003"/>
```

Figure 2.2 header of OpenDRIVE file.

Each OpenDRIVE file contains a header mainly describes the basic information like boundary of the map, name of the file, OpenDRIVE version, etc, like figure 2.3

```
<link/>
<planView>
<geometry
s="0.0000000000000000e+000"
x="0.0000000000000000e+000"
y="0.0000000000000000e+000"
hdg="0.0000000000000000e+000"
length="5.0000000000000000e+001">
```

Figure 2.3 road and plan view.

This part describes the plan view of all roads that are contained in the map. For each road, OpenDRIVE file stores name, ID and length of the road. Each plan view consists of several consecutive geometries

2.3 OpenStreetMap

OpenStreetMap is a free editable map of the whole world. “OpenStreetMap (OSM) follows a similar concept as Wikipedia does, but for maps and other geographic facts”(8). People from all around the world could upload their location data from mainly GPS devices. Of course the uploaded data can be modified and corrected by anybody else who has the good reason, usually facts missing or errors.

Here is the main webpage of OpenStreetMap: http://www.openstreetmap.org/. You can get map information like figure 2.4 shows:

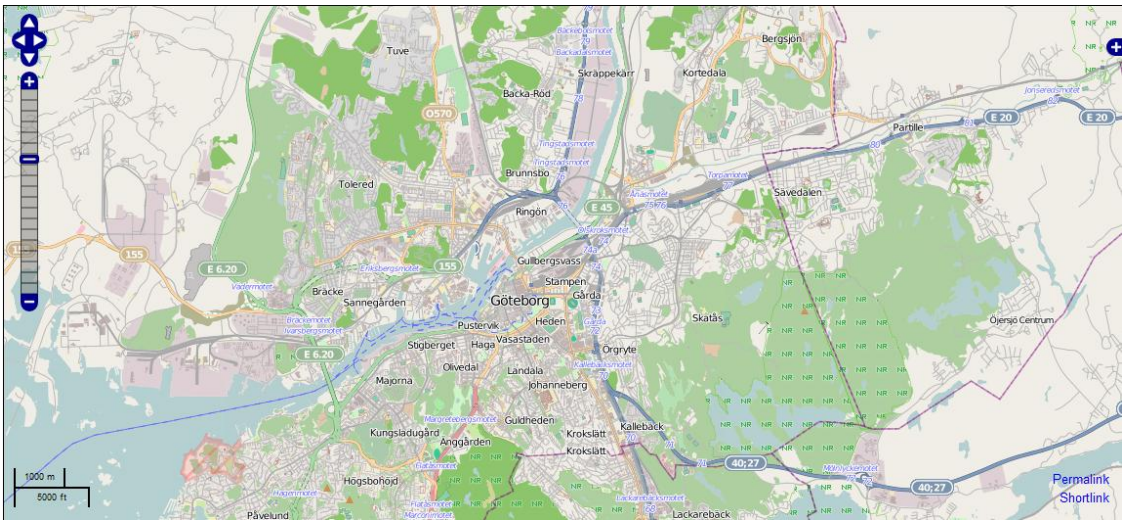

Figure 2.4 Map of Gothenburg exported from OpenStreetMap

Figure 2.4 shows transport information in Gothenburg area, including road, highway, railway, motorway, river, etc. One can export information of this area as PNG format or XML format. What the project use is XML data format.

In the exported XML data format file, each road consists of a series of nodes which are defined as figure 2.5 shows:

```
<node id="306508408" lat="57.7131658" lon="12.5829357" user="bengibollen"
uid="9380" visible="true" version="5" changeset="509067" timestamp="2008-10-
22T22:43:40Z"/>
```

Figure 2.5 information of node that is described in OpenStreetMap file

There are nine attributes for each node, yet what the project uses are first three attributes, i.e. id, lat, lon. Each node has a unique ID number, lat means latitude of this node, lon means the longitude of this node. Obviously all nodes are located by a unique combination of latitude and longitude.

2.4 Bézier Curves

Bézier Curves method, which was developed by the French engineer Pierre Bézier, is highly useful and convenient. So it is widely used in Computer Aided Design systems to build smooth curves, especially graphics packages and drawing/painting packages. Generally, a Bézier Curve section is fitted by a series of control points, Let's say we have n control-point: $p_i = (x_i, y_i)(i = 0,2,...n-1)$, and here we define Bézier Curve of

degree n recursively, it is expressed as a linear interpolation between two Bézier curves of degree n – 1 (9):

Let's define $B_{p1p2..pn}$ as Bézier curve from p_1 to p_n. Then we can get

$B_{k,n}(t) = (1-t)B_{k,n-1}(u) + tB_{k-1,n-1}(t)$, $(n > k >= 1)$.

Then we can construct Bézier curves by this expression, here are some examples of how to construct Bézier curves:

2.4.1 Linear curves

According to the expression, the curve is given by

$B(t) = p_0 + t(p_1 - p_0) = (1-t)p_0 + tp_1$, $t \in [0, 1]$

Obviously it is equivalent to linear interpolation. So B(t) describes a straight line from p_0 to p_1 in this case. Figure 2.6 describes the procedure of generating a linear Bézier curve(10):

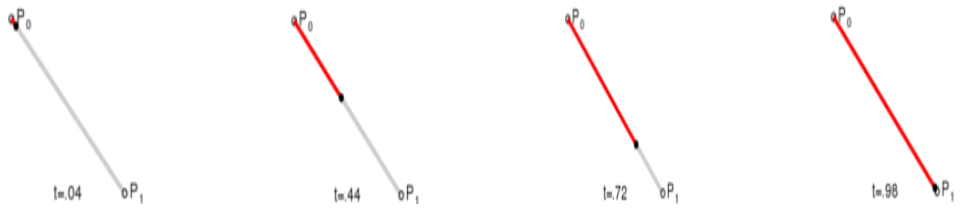

Figure 2.6 linear Bézier curve generation (from http://en.wikipedia.org/wiki/B%C3%A9zier_curve)

2.4.2 Quadratic curves

For quadratic Bézier curves one can construct curves recursively, first get two points q_0 and q_1 in the following way:

q_0 varies from p_0 to p_1

q_1 varies from p_1 to p_2

Clearly both q_0 and q_1 describe a linear Bézier curve, and we also have $q_0, q_1 \in [0, 1]$. Then we can get point B(t) varies from q_0 to q_1 just like linear curves, these series of points generate a quadratic Bézier curve. Figure 2.7 describes the procedure of generating a quadratic Bézier curve.

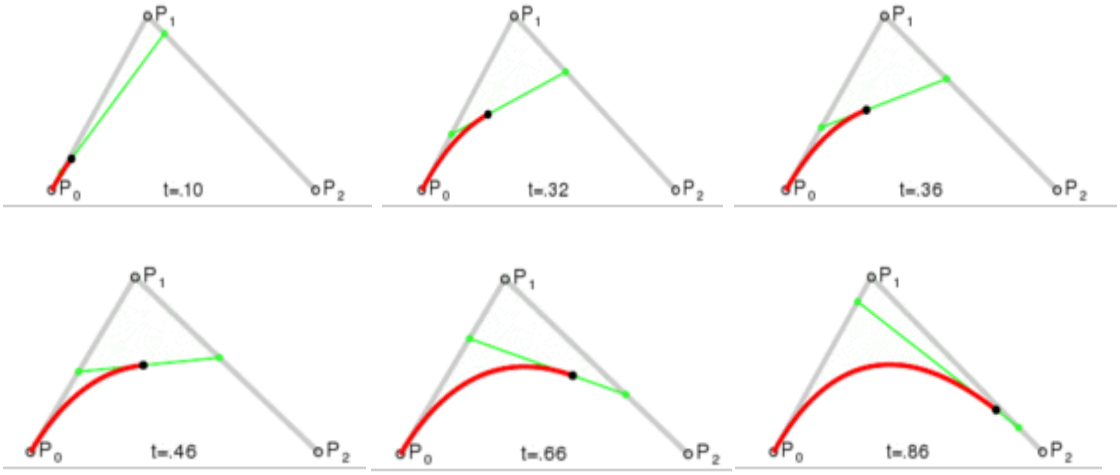

Figure 2.7 quadratic Bézier curve generation (from http://en.wikipedia.org/wiki/B%C3%A9zier_curve)

2.4.3 Higher-order curves

For higher-order curves one needs more intermediate points just like q_0 and q_1, and more recursive layers . The following example describes a fourth-order curves.

First layer: get four points q_0, q_1, q_2 and q_3 in the following way:q_0 varies from p_0 to p_1, q_1 varies from p_1 to p_2, q_2 varies from p_2 to p_2, q_3 varies from p_3 to p_4. Clearly, q_0, q_1 q_2 and q_3 describe a linear Bézier curve.

Second layer:

As said before, q_0, q_1, q_2, and q_3 describe a linear Bézier curve, and we also have q_0, q_1, q_2, $q_3 \in [0, 1]$. Then we can get point $r_0(t)$ varies from q_0 to q_1, $r_1(t)$ varies from q_1 to q_2, $r_2(t)$ varies from q_2 to q_3. $r_0(t)$, $r_1(t)$ and $r_2(t)$ describe quadratic curves.

Third layer:

We can get point $s_0(t)$ varies from r_0 to r_1, $s_1(t)$ varies from r_1 to r_2, $r_0(t)$, $s_0(t)$ and $s_1(t)$ describe quadratic curves. $s_0(t)$ and $s_1(t)$ describe cubic curves.

Fourth layer:

we can get point B(t) varies from s_0 to s_1, these series of points generate a fourth – order Bézier curve. Figure 2.8 describes the procedure of generating a fourth-order Bézier curve.

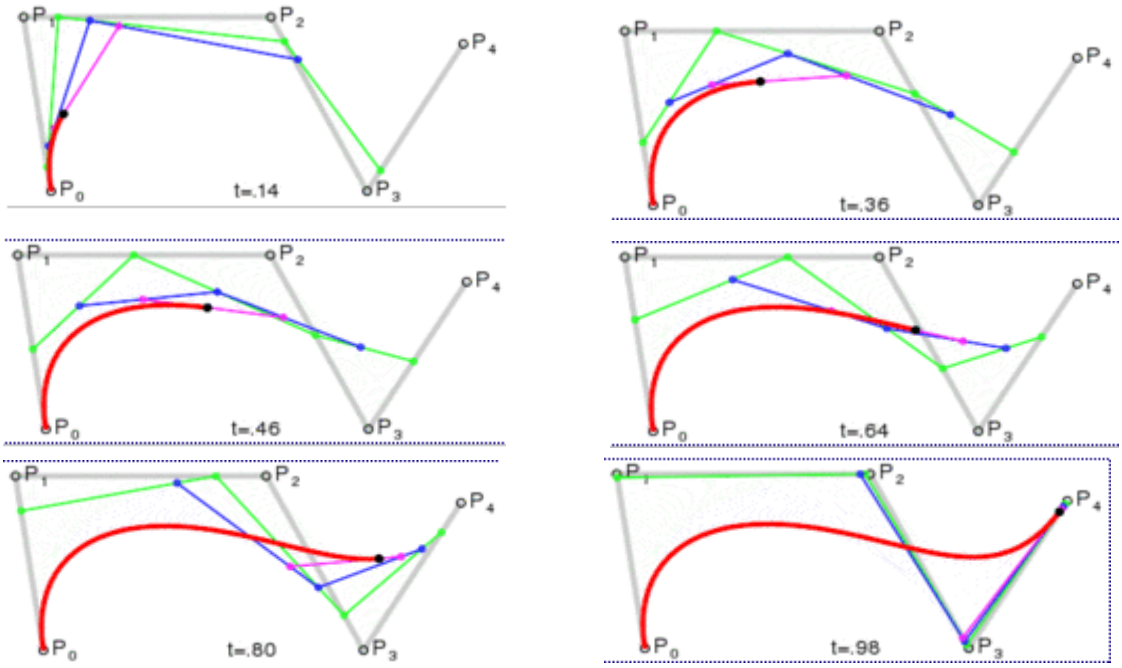

Figure 2.8 fourth-order Bézier curve generation (from http://en.wikipedia.org/wiki/B%C3%A9zier_curve)

2.5 Coordinate systems

2.5.1 Global coordinate system

The global coordinates system is a right-handed coordinate system, its axes point to the following directions:

X: forward

Y: left

Z: up

For geographic reference, the following convention applies

X: east

Y: north

Z: up

2.5.2 Local coordinate system

The local coordinate system is a right-handed coordinate system with the axes pointing to the following directions:

X: forward

Y: left

Z: up

The local system could only be used in space tracing, and one needs to provide the full track coordinates system and orientation of its origin.

2.5.3 Track coordinate system

The track coordinate system applies along the reference line of a road. It is a right-handed coordinate system with the axes pointing to the following directions:

s: position along reference line, measured in [m] from the beginning of the track, calculated in the xy-plane (i.e. not taking into account the elevation profile of the track)

t: lateral position, positive to the left

h: up

Figure 2.9 shows s and t direction.

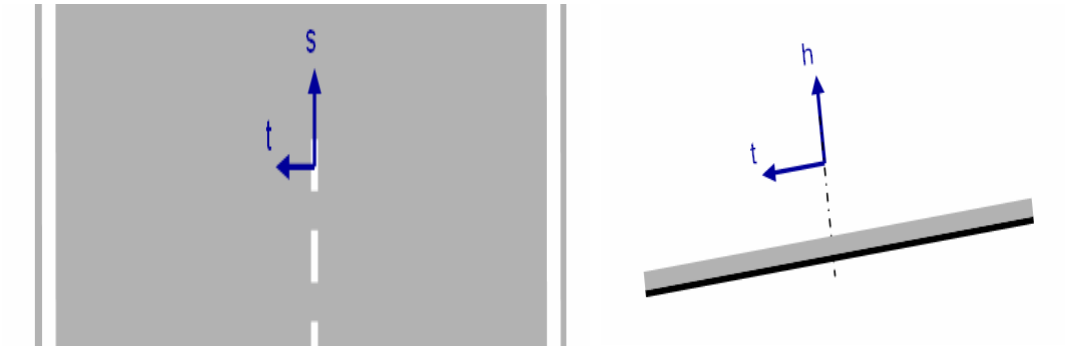

Figure 2.9 fourth-order Bézier curve generation (from OpenDRIVE menu, V 1.3)

2.6 Other solutions

As listed before, there are four kinds of geometries that OpenDRIVE could support: Line, Spiral, Arc, Cubic polynomial. This section evaluates the pros and cons of the other three geometries.

2.6.1 Line

Line describes a straight line as part of the road's reference line. This record has no arguments.

Benefits

- Easy to be set arguments.

Cons

- Obviously almost all roads cannot be described as straight line, so Line is definitely out of the table.

2.6.2 Spiral

Spiral describes a spiral as part of the road's reference line. This record has two arguments.

curvStart: curvature at the start of the geometry.

curvEnd: curvature at the end of the geometry.

For this type of spiral, the curvature changes from curvStart to curvEnd linearly.

Benefits

- Fewer arguments: compare with Cubic poly($y = a + b*x + c*x^2 + d*x^3$, four arguments), Spiral has just two arguments, curvature at the start/end of the element.
- Good performance in some cases: actually many roads are built roughly as spiral because of safety reason. So spiral can perform very well in some certain segments.

Cons

- To many geometries are needed: here are the figures standard of spiral and cubic polynomial:

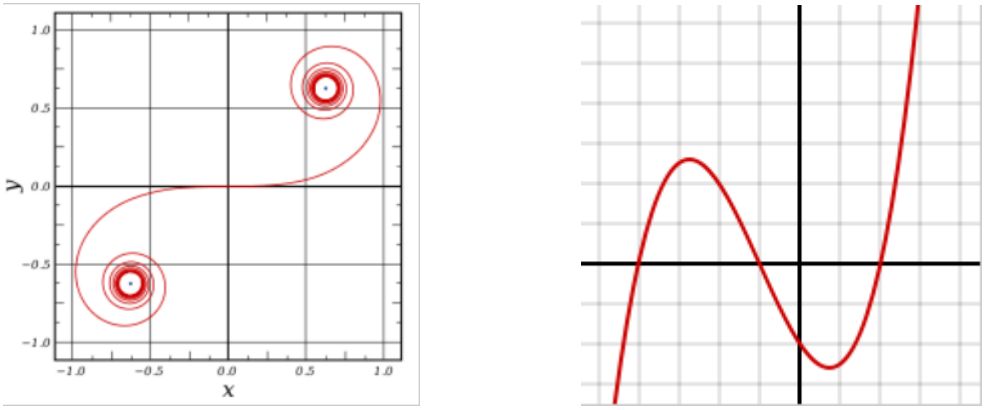

Figure 2.10 standard spiral and cubic polynomial (from http://en.wikipedia.org/wiki/Spiral)

According to the figure 2.10, you can notice that cubic polynomial could describe two curves which spiral cannot, that means you need to separate a long road into much more geometries if you want to describe it with spiral but

not polynomial. The disadvantages of excess geometry will be described in section 3.1

2.6.3 Arc

Arc record describes an arc as part of the road's reference line. This record has just one argument.

curvature: constant curvature throughout the geometry.

Benefits

- Easy to implement: arc has just one argument need to be set, and it could be calculated in an easy way.

Cons

- Bad accuracy: arc could not perform as good as spiral and cubic poly.
- Jerks: arc could lead to jerk between two geometries. Jerk means a sudden change of curvature of road at one point.
  Our previous program was coded with matlab, it chose geometry arcs to describe the road. two consecutive arcs with different curvatures may lead to jerks like figure 2.11:

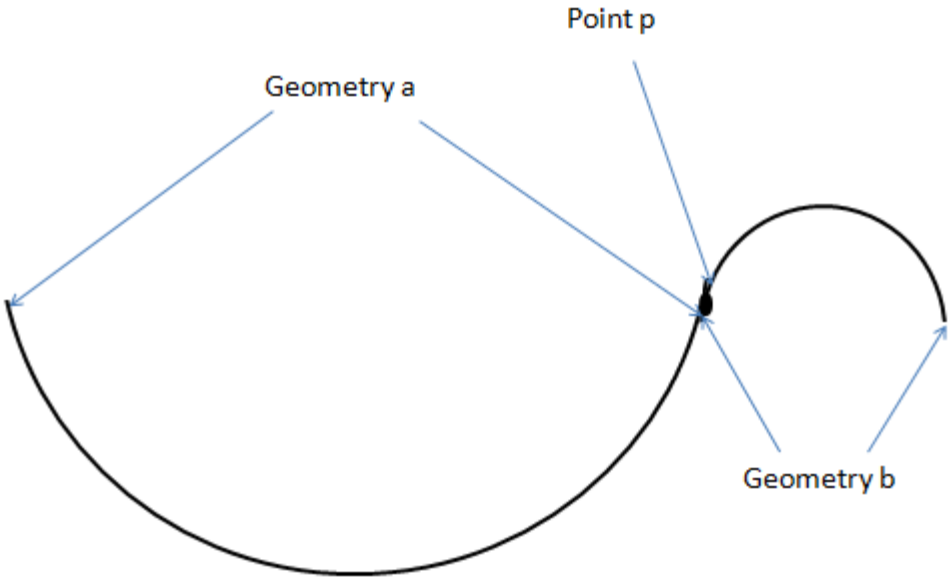

Figure 2.11 jerks between two segments of road

The road shown in the picture above consists of two segments and both of them are described with arcs. However, these two arcs have different radius of curvature, segment a has a very big value of radius of curvature, which means the road is very smooth in segment a, but it turns to steep immediately after point p, i.e. the intersection of segment a and b, which would lead to an

embarrassing problem: driver need to turn the steering wheel abruptly when he drive to the point a because there has a jerk in that place. That is of course not a good phenomenon. Obviously, the road generated by the program should be more continuous so that the driver can drive the vehicle easier. Continuity in here means derivative of the road changes smoothly, and the rate of change of curvature of the road is smooth.

3.Program design

This chapter describes the details of how the program is designed. The most essential part of the program is the conversion from OpenStreetMap data format to OpenDRIVE data format.

3.1 Goals of design

There are some constraints that an effective road generating system must or should comply. In this section, I will review some of the basic and special requirements that the program must implement, and I will also explain how could they have influenced the design choices.

3.1.1 Get OpenDRIVE file

Of course the basic goal of the project is generating OpenDRIVE data format file. The main problem here is a series of nodes defines the layout of the road's reference line in OpenStreetMap format file, but the road in OpenDRIVE format file is described by a sequence of road geometry records. So substantially, what the project needs to implement is generating road's plan view which is defined by a sequence of road geometry records with a series of nodes.

3.1.2 Number of geometries

The file is going to generate a road which is used in the simulator. Then the simulator will do the automatic driving simulation for the car. One important thing is that the simulator needs to run many simulations in parallel. More geometry will increase the processing period dramatically, so we need to try to minimize the number of geometry.

On the other hand, we also need to try to get a precise result, i.e. minimize the error. The project uses cubic polynomial to describe the geometry, i.e. y = a*x^3 + b*x^2 + c*x + d. Since the expression has four coefficients, so a cubic polynomial could hit any four points with no error. Suppose there is a road which consists of n nodes in OpenStreetMap file. Therefore, technically no extra errors would be introduced if the project describes the road with n/4 geometries in OpenDRIVE format. But as mentioned before, the number of geometries is clearly so big that do not fulfill our goals of design. So we need to find the balance point between geometry limitation and error control.

3.1.3 Error Control

The previous project that was developed with matlab did an experiment and generated a 14km road. There was an error of around 400 meters at the end of the road, which is intolerable for the simulator. For a long road, some small rounding deviation at the beginning of the road may lead to very big error at the end of the road. Instead of a big error occurred at the end of the road, it is better for the project to separate the big error at the end of the road into smaller one and assign them to each segment of the road evenly. The second step that the project needs to do is trying to minimize the error in each segment.

3.1.4 Smoothness

As mentioned before, road generated from our previous program has jerks at the junction point of two geometries. The new program should try to avoid this problem.

3.2 Read/Write

3.2.1 Overview

The first step that the program needs to do is reading information from OpenStreetMap file, mainly attributes of nodes, then store them into structure of C++. Of course the last thing should be finished is generating the OpenDRIVE file with all the attributes that are calculated by the project. Since both OpenStreetMap data source and OpenDRIVE format are XML data format, so the program used tinyXML package to process these file. “TinyXML is a small C++ XML parser, it can be easily integrating into other programs, and it is a simple, stable, basic XML parser used by many open source and commercial products” (11).

3.2.2 Read OpenStreetMap file

First of all, the program should read information of all the ways which are stored in OpenStreetMap file. The data structure of way in C++ is defined in figure 3.1:

```
struct Way
{
        int ID;                         //ID of way
        vector<int> infoOfNodes;        //all the ID of nodes which are
                                        //composed of this road.
        string user;                    //name of user;
        int UID;                        //ID of user
        bool visible;                   //visibility of way
        int version;                    //current version
        int changeset;
        string timestamp;               //timestamp of this road
        string key;
        string value;                   //mainly consists of motorway,
road etc. one level lower then key.
};
```

Figure 3.1 road information in C++ data structure

The most important member in this structure is *infoOfNodes*. This stores all the nodes which are in this road. One can easily trace the whole road with this member.

Secondly, read information of nodes which are composed of every road respectively. The data structure of node is shown in figure 3.2:

```
struct Node
{
        int ID;                         //ID of node
        double lat;                     //Latitude;
        double lon;                     //longitude;
        string user;                    //name of user;
        int UID;                        //ID of user
        bool visible;                   //visibility of node
        int version;                    //current version
        int changeset;
        string timestamp;               //timestamp of this node
        int x, y;                       //x,y coordinates
};
```

Figure 3.2 node information in C++ data structure

As said before (2.3), the most important three members are ID, lat and lon.

3.2.3 Write OpenDRIVE file

After finish calculating, the program should generate a OpenDRIVE file according to the result, mainly information of geometries. The structure of geometry in C++ is defined in figure 3.3:

```
struct geometry
{
        double x, y;                    //starting point
        double hdg;                     //starting orientation
        double s;                       //start position (track-
                                        coordinate system)
        double length;                  //length of the geometry
        int startPoint;                 //startPoint of the geometry
        int endPoint;                   //endpoint of the geometry
        int numOfNodes;                 //end - start + 1;
        parameter par;                  //parameter of cubic poly
};
```

Figure 3.3 geometry information in C++ data structure

Geometry in OpenDRIVE data format is described with:

```
<geometry s="" x="" y="" hdg="" length="">
```

One can generate all attributes with members in the structure.

3.3 Road separation

As mentioned in section 3.1.2, we need to find the balance point between geometry limitation and error control, then we can easily find out that how to separate road is a very important problem.

3.3.1 Lat/Lon conversion

There are three coordinate systems that are described in OpenDRIVE data format, but nodes in OpenStreetMap file are located by Latitude/Longitude coordinates system that we cannot use directly in OpenDRIVE, so we need to convert it to global coordinates system firstly, i.e. XY-Coordinates system. Since the roads in Sweden are usually no longer than 500 km, so the error that may arise in the conversion process should be very small, and what our program needs to get is nothing to do with high absolute accuracy but relative accuracy. Therefore, we can just use two simple formulas to do the conversion:

y = lat_Radian * (40007000 / (2.0 * PI))

x = lon_Radian * (40075000 / (2.0 * PI)) * cos(lat_Radian)

These two formulas simply regard the whole earth as the global coordinates system. lat_Radian and lon_Radian are latitude and longitude respectively, x and y are X and Y in global coordinates system. The length of equatorial radius is 6378.137km, and the length of polar radius is 6356.752km, so it could easily calculate that the equatorial circumference of the earth is roughly 40075000 meters and the polar circumference of the earth is roughly 40007000 meters. One can regard the earth as figure 3.4 by using the formulas above.

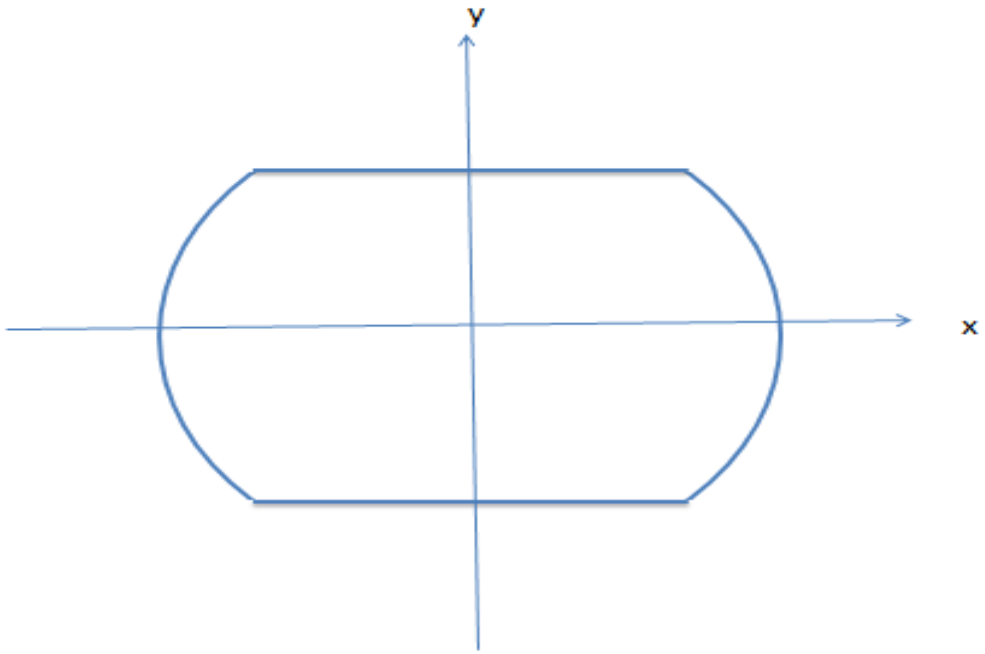

Figure 3.4 earth in global coordinates system.

3.3.2 Error evaluation

According to the goals of design for the program, the road shall be separated based on two rules:

1. Each segment of the road shall be described by cubic polynomial while the introduced error is as small as possible.

2. The number of segments should be as few as possible.

This is a very difficult problem since a smaller error means the road should be separated to more segments, meanwhile less segments will of course introduce more error. It is very hard to find the balance point between these two rules. The method that the project uses now is (12):

1. Start a new segment, for the first n (n could be any number) nodes in this segment, do least squares fitting and store the value of error.

2. Introduce n more nodes, do least squares fitting for all the nodes in this segment again and compare the error with the last one.

3. Stop adding new node if the error has a dramatic increase, then start a new segment and do step one again, or stop the algorithm if all the nodes have already been processed. If the error increases gently, store the new error and do step two again.

This is a greedy algorithm, which could find a relatively good result but not optimal result. Since this problem may not have an optimal result, (we do not know how much error could "equal" to one more segment) and this algorithm could be easily designed and implemented. So we decided to use this algorithm to solve problem.

3.3.3 Coordinates conversion

Every cubic polynomial is described in the unique local u/v coordinate system of the starting point. For each local coordinate system, starting point is the origin of coordinates, u pointing in the s direction in track coordinate system and v pointing in the t direction in track coordinate system. Each local coordinate is calculated by $v_{local} = a + b*du + c*du^2 + d*du^3$.

"The conversion of u/v co-ordinates into x/y co-ordinates can be performed easily by simple geometric transformations, (i.e. one translation and one rotation) relative to the starting point" (13), as shown in figure 3.5:

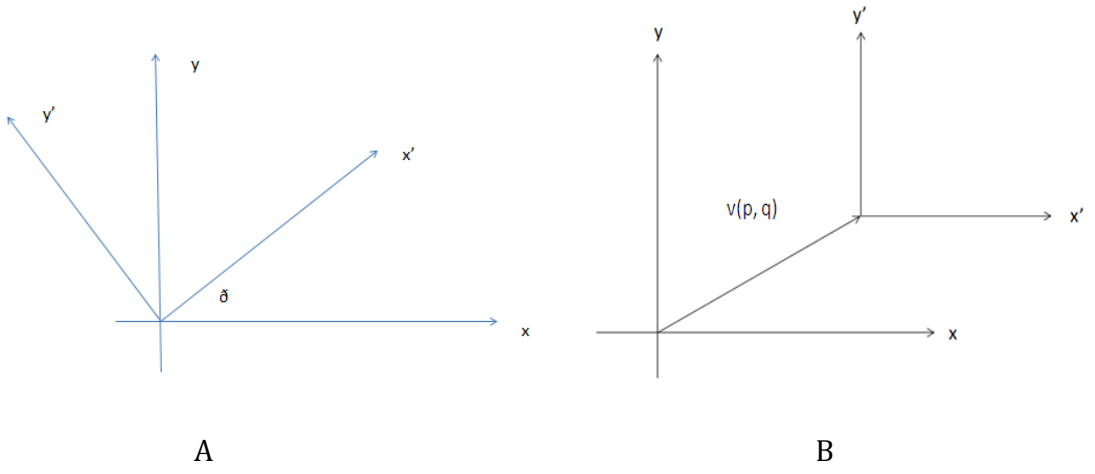

Figure 3.5 coordinate transformations

In figure 3.5A, if we want to do transformation from xy-coordinates system to x'y'-coordinates system, we need to do rotation for the angle ð. The formula is:

$x' = x\cos(ð) + y\sin(ð)$

$y' = -x\sin(ð) + y\cos(ð)$

In figure 3.5B, if we want to do transformation from xy-coordinates system to x'y'-coordinates system, we need to do translation for the vector v(p, q). The formula is:

$x' = x + p$

$y' = y + q$

So if we do translation and rotation respectively, the formula is:

$x' = x * \cos(ð) + y * \sin(ð) - p * \cos(ð) - q * \sin(ð)$

$y' = -x * \sin(ð) + y * \cos(ð) + p * \sin(ð) - q * \cos(ð)$

3.4 Geometry calculation

This is the most important part of the program. According to the goals of design, the project needs to separate the error at the end of the road is not that big. That requires the curve of geometry should hit the starting and ending points for each segment of the road. In this way one can ensure that the error in current segment would not be brought to the next segment.

All attributes of the geometry shall be calculated. The geometry information in OpenDRIVE format is split into a header which is general to all geometric elements and a subsequent bead which contains the actual geometric element's data (depending on the type of geometric element, in our project it is cubic polynomial, so the element's data are a, b, c, and d).

There are four attributes for each geometry header:

s:	length from beginning
x, y:	global coordinates of the starting point
hdg:	start orientation of geometry
length:	length of geometry

And there are four attributes for each subsequent bead:

a: constant coefficient

b: linear coefficient

c: quadratic coefficient

d: cubic coefficient

The first two attributes of geometry header could be easily calculated, length from beginning means length from the first node of the whole road to starting node of the current geometry. It is equal to the sum length of geometries before the current one. Attribute x and y indicates has already been stored in structure nodes (in section 3.3.3). However, we need to find the corresponding method to calculate the other attributes.

3.4.1 Start orientation

This attribute indicates the start orientation of the current geometry in local coordinate system, i.e. v-axis in local coordinates system. According to the definition, the angle contained by local coordinates system and global coordinates system is the start orientation. According to the goals of design, it needs to be set in a reasonable way so that no jerk would be introduced at the junction point of two geometries.

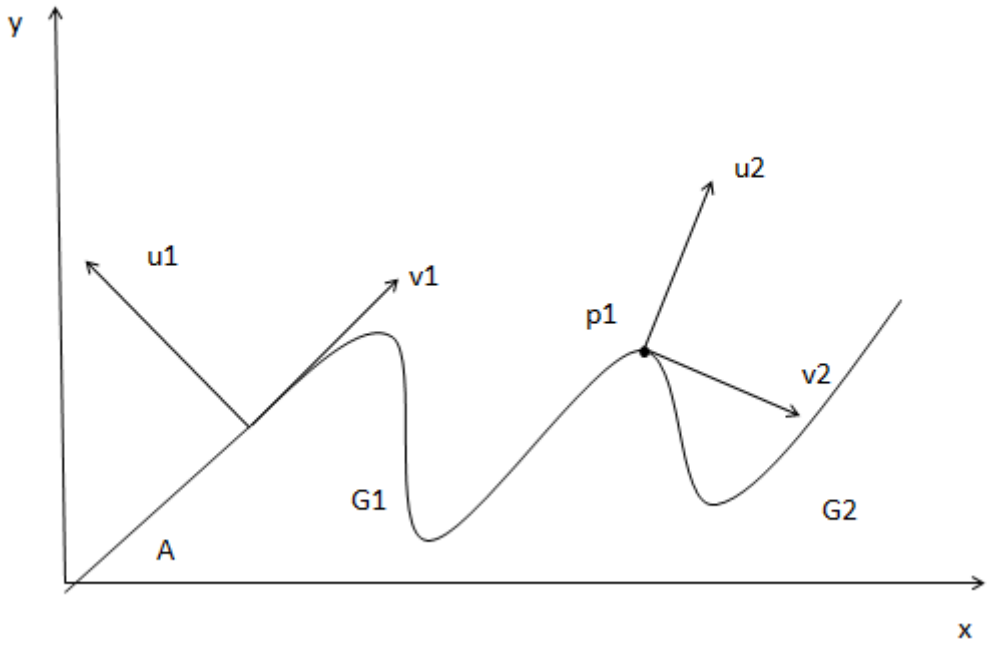

Figure 3.6 start orientation of G2

In figure 3.6, p1 is the ending point of geometry G1, it is also the starting point of G2. u_1v_1-Coordinates system and u_2v_2-Coordinates system in the figure are local

coordinates system of geometry G1 and G2 respectively. v_1-axis and v_2-axis are start orientation of G1 and G2. In the program, the direction of v_2-axis is defined by the derivative of G1 in point p1 in u_1v_1-Coordinates system. The second geometry will start with the same direction of the first geometry ended if we define starting orientation in this way. It could ensure the road tracing continuously, especially at the junction point between two geometries.

Suppose expression of G1 is $v_1 = a_1 + b_1*u_1 + c_1*u_1^2 + d_1*u_1^3$,and p1 in G1' s local coordinates system locates at (s, t). Firstly we derivation of the expression, get $v_1' = u_1 + 2*c_1*u_1 + 3*d_1*u_1^2$. Then we can get the derivative in point p1 by inserting (s, t): derivative = $u_1 + 2*c_1* s + 3*d_1* s^2$. However, this derivative is in u_1v_1-Coordinates system, we need to convert it into the global coordinates system, since the angle contained by u_1v_1-Coordinates system and global coordinates system is A, so we can easily get orientation of v_2 = arc tan (derivative) + A.

In conclusion, start orientation of the current geometry is equal to arc tan of derivative plus start orientation of previous geometry.

One interesting thing is for each $y = a + b*x + c*x^2 + d*x^3$, we have $y' = b + 2*c*x + 3*d*x^2$, the derivative at the starting point (0, 0) is always b, it is like a static potential start orientation for cubic polynomial, which could also lead to jerks between two segments. The solution of this problem will be described in 3.4.3.

3.4.2 Length of geometry

This attribute indicates the length from the starting point to ending point of the geometry, there is unfortunately no close form expression for the arc length of a cubical polynomial. One resort is to do numerical integration of the expression, i.e. y= a * x^3 + b * x^2 + c * x + d.

Length of curve = $\int_a^b f(x)dx = \int_a^b \sqrt{1+(\frac{dy}{dx})^2}dx$

y' = 3a*x^2 + 2b*x + c.

So we have Length of curve = $\int_a^b \sqrt{1+(3a*x^2+2b*x+c)^2}dx$

Unfortunately, we do not have such a ready formula that can solve fourth-degree integration, so the project uses midpoint rule of numerical integration to solve it.

Suppose there is a continuous function f(x), x is the variable. The goal of the Midpoint rule is computer ∫f(x)dx over interval from a to b. Estimate integral by M = (b-a)*f(c), where c=(a+b)/2 According to the midpoint rule, firstly the curve shall be separated into n segments with the same length h, like figure 3.7:

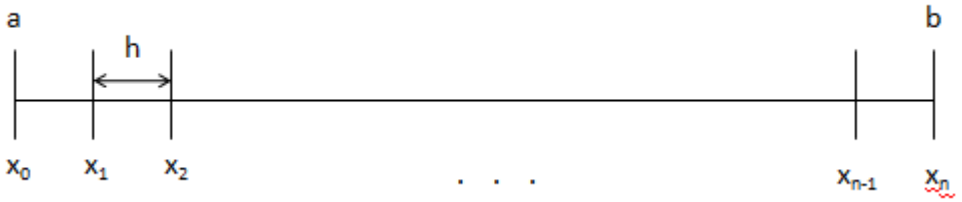

Figure 3.7 numerical integration of midpoint rule

Length of curve $= \int_a^b f(x)dx = h\left[f(\frac{x_0+x_1}{2}) + f(\frac{x_1+x_2}{2}) + \ldots + f(\frac{x_{n-1}+x_n}{2}) \right] + R_T = M_n + R_T$

In the formula: $h = \frac{b-a}{n}$ $x_k = a + \frac{k}{n}(b-a)$ $R_T = \frac{(b-a)h^2}{24} f''(\varepsilon), a < \varepsilon < b$

R_T is the truncation error (14).

3.4.3 Coefficients

This attribute indicates coefficients of cubic polynomials, i.e. a, b, c, d in expression y = a + b*x + c*x^2 + d*x^3. As mentioned before, the project needs to construct a curve (here this curve is a mathematical function of cubic polynomials) , that has the best fit to a series of nodes, and it could be solved by least squares fitting methods. There are several mathematical packages which can be used to implement least squares fitting problem, like: Levenberg-Marquardt(15), ALGLIB, MINPACK(16), etc.

ALGLIB is a cross-platform numerical analysis and data processing library which is used in the project. There are some good reasons that we choose this package for our project:

1. It has an easy to use linear least squares fitting application package that could be used conveniently in the project.

2. It supports C++ programming language and Windows operating system, our project is coded by C++ and run in Windows operating system,

3. It supports least squares fitting with constraints, as mentioned before, our fitting case has two constraints, i.e. each curve of geometry needs to hit the starting point and ending point perfectly.

The package contains a function ”lsfitlinearc” for solution of the constrained linear least squares problems. The function is defined as (17):

lsfitlinearc(Y, Fmatrix, Cmatrix, N, M, K , info, C, Rep)

Definition of input parameters:

Y: array[0..N-1] Function values in N points.

Fmatrix: a 2D matrix of basis functions values, array[0..N-1][0..M-1]. FMatrix[i, j] means value of jth basis function in ith point.

Cmatrix: a 2D matrix that is used to define constraints, array[0..K-1][0..M]. ith row of CMatrix corresponds to ith linear constraint, the constraint is defined in this way: CMatrix[I][0]*C[0] + ... + CMatrix[I][M-1]*C[M-1] = CMatrix[I][M]

N: number of points that are used. N = {1, 2, 3,…. ∞}.

M: number of basis functions, M = {1, 2, 3,…. ∞} .

K: number of constraints, K = {1, 2, 3,…. M}.

Definition of output parameters:

Info: return error code.

C : a linear matrix that is used to store decomposition coefficients, array[0..M-1]

Rep: return all types value of error.

Suppose we have n nodes in one segment, (x_1, y_1), (x_2, y_2),… (x_n, y_n), then we have:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ . & . & . & . \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ y_n \end{bmatrix}$$

Yet according to the 3.4.1, each cubic polynomial has a potential static start orientation of b, so here we always set b to 0, then no jerks will be introduced at the junction node. Therefore, we can form matrix as follows:

$$\begin{bmatrix} 1 & x_1^2 & x_1^3 \\ 1 & x_2^2 & x_2^3 \\ . & . & . \\ 1 & x_n^2 & x_n^3 \end{bmatrix} \begin{bmatrix} a \\ c \\ d \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ y_n \end{bmatrix}$$

One can get coefficients a, b, c, d from output parameter C, and get value of average error from output parameter Rep.AvgError.

3.5 Summary of conversion process

3.5.1 Goals of design

There are four goals for the program designing.

1. Generate OpenDRIVE file with OpenStreetMap data source.

2. Try to control the number of geometry separated by the road.

3. Try to control the error appeared in the fitting process.

4. Generate the road as smooth as possible.

3.5.2 Road separation

Before geometry calculating, the program should separate a long road into several geometries.

1. Do conversion from lat/lon system to global coordinates system.

2. Separate road into geometries according to a greedy algorithm.

3. Do conversion from global coordinates system to local coordinates system for each geometry.

3.5.3 Geometry calculation

There are eight attributes should be calculated, four of them are in the header of geometry.

1. s: start position in s-coordinate, could be easily calculated by summing previous length of geometries.

2. x, y: start position in xy-coordinates system, could be easily set by information of node stored in the structure.

3. hdg: start orientation. It is defined by the derivative of last geometry at the ending point.

4. length: length of the geometry's reference line. It is calculated by numerical integration of midpoint rule.

The other four attributes are in the subsequent bead, for the cubic polynomial they are a, b, c, d, i.e. coefficients of expression. These attributes are calculated by least squares fitting method.

4.Results

This chapter assessed how well the solutions I have developed are performing on the data sets, i.e. OpenStreetMap data source. Since the focus of the work has been to how the error it is, so there are two ways that can test my program.

4.1 Datasets

To measure how well the program performs in terms of accuracy, we have exported several groups of data sets and have done some tests for our program. This is the typical one: The data set consists of a road of roughly 1.5 km from OpenStreetMap, and the road consists of 372 nodes.

4.2 Testing environment

The entire test was completed on a computer with the following specifications:

- Processor: AMD XP +4800
- Memory: 2Gb
- Drives: A 150 GB 10 000 rpm Western Digital Raptor and four 300GB 7200 rpm Seagate Barracuda in RAID 1 +0 association.
- Operating system: Windows 7 SP1
- C++: Visual Studio 2010
- ALGLIB 3.4.0
- GNUPlot 4.4.3

4.3 Test Result

We have designed the following way that can test our project:

1. Do conversion from Lat/Lon system to global coordinates system for every node in OpenStreetMap. One can regard this series of nodes as original nodes.

2. Get OpenDRIVE file by our program.

3. Trace back the road described in OpenDRIVE file and get series of nodes with this road.

4. Try to plot the road with nodes in step 1 and 3 separately, then check the difference between these two roads.

GNUPlot is a portable driven graphing tool, it is an easy using open source software that can plot curve by a series of consecutive nodes. We have plotted the original nodes by GNUPlot, the result looks like Figure 4.1 shows:

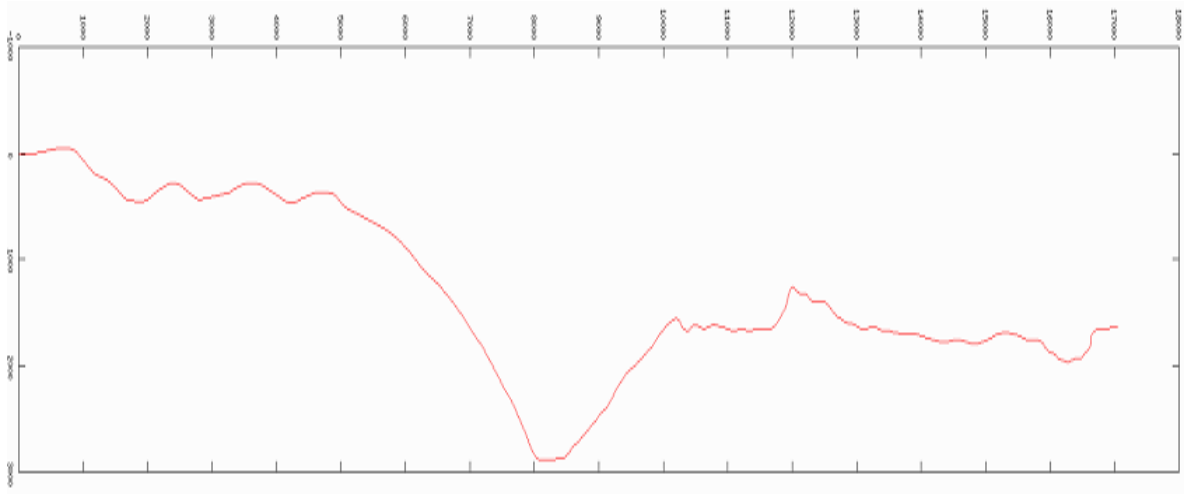

Figure 4.1 curve of the original road.

The unit of x and y axes is meter, the scale of each slot is 1000 meters.

As mentioned before (in section 3.3.2), we have designed an algorithm that is used to separate the road. Firstly, we try to set n in step 1 to 20, then run the program and generate OpenDRIVE file. There are 8 geometries in this file. Then we plot the original road and generated road at the same time.

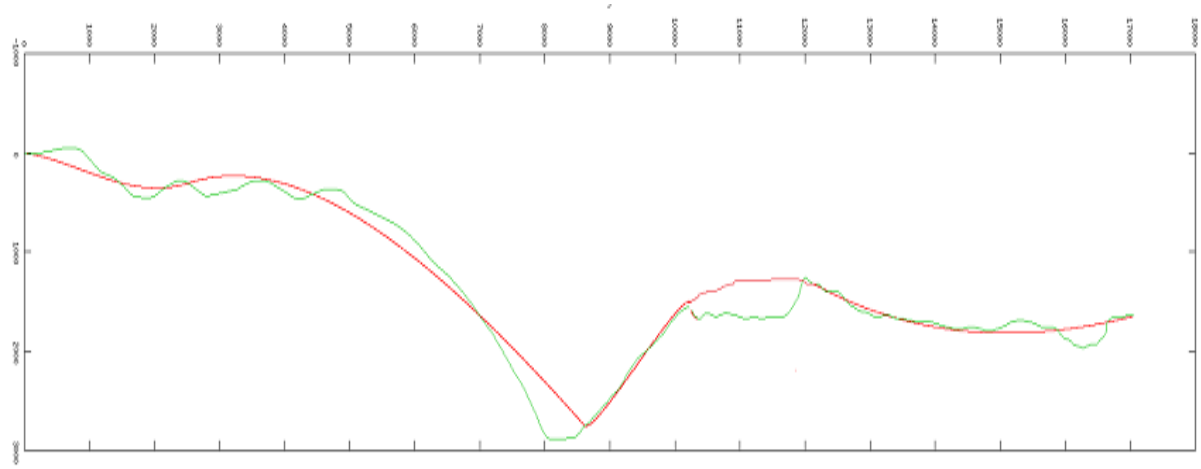

Figure 4.2 curve of the original road(green) and generated road(red) n = 20.

geometry	Error(meter)	Length(meter)
1	34.1701	2555.7599
2	90.8312	6745.9264
3	40.275	2175.0050
4	147.096	2799.6547
5	69.5064	4979.8234
average	76.37574	3851.2336

Table 4.1 error of each segment (n = 20)

As shown in figure 4.2 and table 4.1, one can see there is rather big error in every geometry, especially for the last part of the road.

Now we set n to 2 this time, then run the program and generate OpenDRIVE file. There are 92 geometries in this file. Then we plot the original road and generated road at the same time.

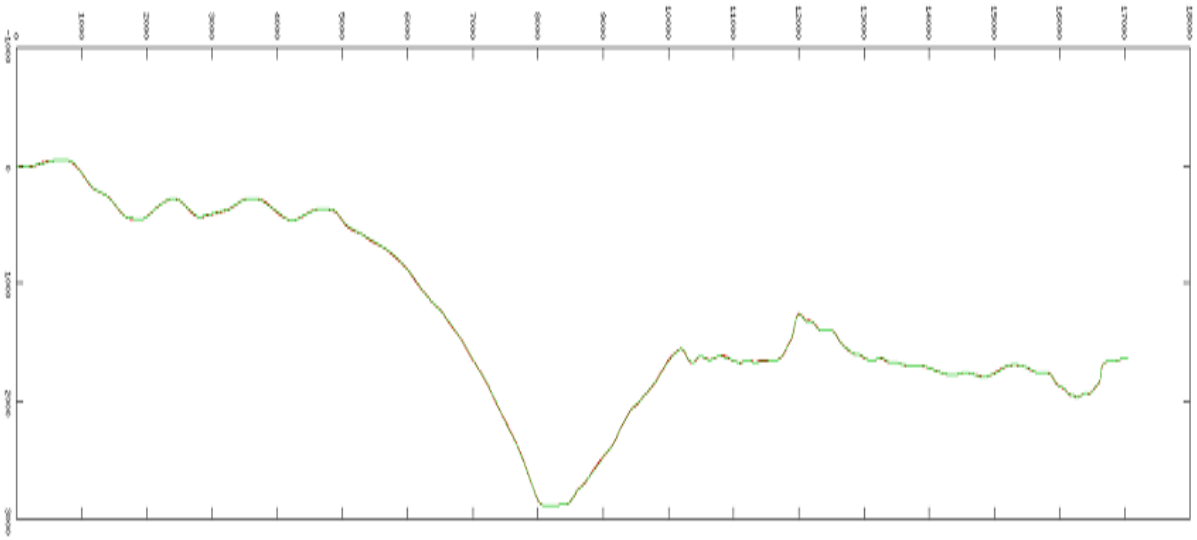

Figure 4.3curve of the original road(green) and generated road(red) (n = 2).

geometry	Error(meter)	Length(meter)
1	0.672154	330.8815
2	1.6795	401.0149
3	1.28874	260.2727
4	0.848396	260.5455

…	…	
89	0.654744	96.0104
90	0.330509	85.5145
91	0.814035	120.2601
92	0.615895	187.9328
average	0.968477	209.6592

Table 4.2 error of each segment (n = 2)

As shown in figure 4.3 and table 4.2, the original road and generated road nearly share the same curve, and the error in each segment is also very small.

We did some more tests for our program. Exactly same as what we have predicted, if we want to get an accurate result, one needs to separate the road into more segments. For test case 1, the average error for all segments is roughly 76 meters, which means the road's general properties could be used. But if we want to try to use this data together with model of the environment then we must get closer to the resolution of the GPS system < 10 meters, and sometimes we need to get even more close. That forces us to separate the road like test case 2.

In test case 2, the average error is just 0.968477 meter, and no error of segment is greater than 6 meters in all of my tests. Yet the program generates 92 geometries for this 1.5 km road, this number is much bigger then test case 1. However, it is tolerable with our current simulator.

We have used the road in simulator with the OpenDRIVE file generated by our program, the road is smooth and continuous at the junction point of the geometries, the problems in our last program do not appear in the current version of program. This program fulfills all the requirements we have listed before: error size is small, geometry number is acceptable, the generated road is smooth. So we can draw the conclusion that this is a good way to generate OpenDRIVE file and VTI could get what they want with the current program.

5.Future improvements

Below is a list of potential efficiency improvements in the system.

5.1 Road Separation

The algorithm that we are using to separate road is rather simple, it could find a relatively good solution, but there are definitely some other algorithms that can improve accuracy of result or time efficiency. One can improve the program by updating this algorithm.

For the current algorithm, one can improve it by doing more tests to set n to a reasonable number, so that the program could find the best balance point between error control and geometry number control.

5.2 Junction

Junction is a place where multiple roads meet. It is the most important information of map except road tracing. The junction is one kind of feature that has possibility to be automatically generated based on the current GPS data. One can do road tracing for all the roads in the map and try to detect where the two roads meet. This is also a complex task and its workload is almost equal to the current project, since this is a six months project so I do not have enough time to implement this part.

5.3 Other features

There are also some other features which cannot be get with the OpenStreetMap data format, like overpasses, roundabouts, road lines, elevation and road textures. Since the only useful information of nodes in OpenStreetMap is Lat/Lon coordinate, for another word, it is a two dimensional system data format. But clearly we need three dimensional coordinates system to generate overpasses and roundabouts elevation. Of course one needs more information like colour to generate road lines and road textures, but not just by the location of nodes. We need another kind of GPS data format to implement these features.

5.4 Bézier Curve fitting

Bézier Curve fitting is one kind of method which is used to build smooth curve. One can get a smoother road by doing Bézier Curve fitting for the nodes instead of least squares fitting, with a cost of larger error. In practice, a smooth road may be more important than accuracy.

6.Conclusion

During my thesis working, I have finished the following tasks:

1. Found a suitable GPS data source for road generation in simulator.

2. Successfully got OpenDRIVE file with GPS data source.

3. Fixed some bugs in previous version of program. The result is much better than before.

4. Controlled the number of segments of the road in a tolerable level.

In the program, we have solved the following problems:

1. Found a reasonable way (least squares method) to do the conversion between nodes and geometries.

2. Designed an algorithm for the road separation. It could give consideration to both error control and geometry number control.

3. Found a way that can avoid jerks when generating the road.

4. Found ways to calculate the arguments of geometry, especially “hdg”, “length” and coefficients of cubic polynomial.

References

1. OpenDrive specification. (20 June 2011) http://en.wikipedia.org/wiki/OpenDRIVE_(specification). (4 October 2011).

2. Least squares . (30 September 2011) http://en.wikipedia.org/wiki/Least_squares . (04 October 2011) .

3. total least squares. (23 September 2011) http://en.wikipedia.org/wiki/Total_least_squares (04 October 2011).

4. Partial Least squares. (4 September 2011) http://en.wikipedia.org/wiki/Partial_least_squares (04 October 2011).

5. 王福昌，曹慧荣，朱红霞。(2009) 经典最小二乘法与全最小二乘法及其参数估计。 统计与决策. Issue 1, P. 16-17.

6. OpenDRIVE . (27 May 2011) http://www.opendrive.org/projectBackground.htm (04 October 2011)

7. OpenDrive specification. (20 June 2011) http://en.wikipedia.org/wiki/OpenDRIVE_(specification). (4 October 2011).

8. OpenstreetMap. (30 September 2011) http://wiki.openstreetmap.org/wiki/Beginners%27_guide. (04 October 2011)

9. Donald Hearn, M.Pauline Baker. (1997)computer graphics: C version. First edition. p.327-329. Tsinghua University Press

10. Bézier curve. (28 August 2011) http://en.wikipedia.org/wiki/B%C3%A9zier_curve (4 October 2011)

11. TinyXML (20 Mar 2010) http://www.grinninglizard.com/tinyxml/ (4 October 2011)

12. Thomas H. Cormen Charles E.Leiseron Ronald L.Rivest Clifford Stein. (2009) Introduction to algorithms Third edition. MIT Press.

13. Marius Dupuis e.a. (2010) OpenDRIVE menubook. P. 31-33.

14. Lennart Rade, Bertil Westergren. (1993) BETAMathematics Handbook. Second Edition P. 342-343. Lund(Sweden), Studentlitteratur.

15. Kenneth Levenberg (1944) A Method for the Solution of Certain Non-Linear Problems in Least Squares. The Quarterly of Applied Mathematics 2: 164–168

16 MINPACK http://devernay.free.fr/hacks/cminpack/cminpack.html (22 October 2011)

17. ALGLIB Group ALGLIB Reference Manual . http://www.alglib.net/translator/man/ (04 October 2011)